Package: trust (via r-universe)

May 12, 2026

Version 0.1-9

Date 2026-02-10

Title Trust Region Optimization

Depends R (>= 4.2.0)

Imports stats

ByteCompile TRUE

Description Does local optimization using two derivatives and trust regions. Guaranteed to converge to local minimum of objective function.

License MIT + file LICENSE

URL http://www.stat.umn.edu/geyer/trust/

NeedsCompilation no

Author Charles J. Geyer [aut, cre] (ORCID: <https://orcid.org/0000-0003-1471-1703>)

Maintainer Charles J. Geyer <geyer@umn.edu>

Repository https://cran.r-universe.dev

Date/Publication 2026-02-11 08:40:02 UTC

RemoteUrl https://github.com/cran/trust

RemoteRef HEAD

RemoteSha 2eef2538b124efd9a28e838f78d4f7a6e7b61920

Contents

trust *Non-Linear Optimization*

Description

This function carries out a minimization or maximization of a function using a trust region algorithm. See the references for details.

Usage

```
trust(objfun, parinit, rinit, rmax, parscale,
    iterlim = 100, fterm = sqrt(.Machine$double.eps),
    mterm = sqrt(.Machine$double.eps),
    minimize = TRUE, blather = FALSE, ...)
```

Arguments

objfun	an R function that computes value, gradient, and Hessian of the function to be minimized or maximized and returns them as a list with components value, gradient, and hessian. Its first argument should be a vector of the length of parinit followed by any other arguments specified by the ... argument. If the domain of the objective function is not the whole Euclidean space of dimension length(parinit), then objfun should return list(value = Inf) when given a parameter value not in the domain of the objective function and minimize == TRUE. Similarly, it should return list(value = - Inf) when given a parameter value not in the domain and minimize == FALSE. Conversely, when given a parameter value in the domain, it must return a list with components with components value, gradient, and hessian. that are all finite and are the value, gradient, and Hessian of the objective function at the given point. **Warning:** The feature of allowing infinite values to indicate a restricted domain does not allow for true constrained optimization. The algorithm will converge to solutions on the boundary very slowly. (See details below.)
parinit	starting parameter values for the optimization. Must be feasible (in the domain).
rinit	starting trust region radius. The trust region radius (see details below) is adjusted as the algorithm proceeds. A bad initial value wastes a few steps while the radius is adjusted, but does not keep the algorithm from working properly.
rmax	maximum allowed trust region radius. This may be set very large. If set small, the algorithm traces a steepest descent path (steepest ascent, when minimize = FALSE).
parscale	an estimate of the size of each parameter at the minimum. The algorithm operates as if optimizing function(x, ...) objfun(x / parscale, ...). May be missing in which case no rescaling is done. See also the details section below.
iterlim	a positive integer specifying the maximum number of iterations to be performed before the program is terminated.

fterm	a positive scalar giving the tolerance at which the difference in objective function values in a step is considered close enough to zero to terminate the algorithm.
mterm	a positive scalar giving the tolerance at which the two-term Taylor-series approximation to the difference in objective function values in a step is considered close enough to zero to terminate the algorithm.
minimize	If TRUE minimize. If FALSE maximize.
blather	If TRUE return extra info.
...	additional arguments to objfun.

Details

See Fletcher (1987, Section 5.1) or Nocedal and Wright (1999, Section 4.2) for detailed expositions.

At each iteration, the algorithm minimizes (or maximizes) the two-term Taylor series approximation

$$m(p) = f + g^T p + \frac{1}{2} p^T B p$$

where f, g, and B are the value, gradient, and Hessian returned by objfun when evaluated at the current iterate, subject to the constraint

$$p^T D^2 p \leq r^2$$

where D is the diagonal matrix with diagonal elements parscale and r is the current trust region radius. Both the current iterate x and the trust region radius r are adjusted as the algorithm iterates, as follows.

Let f^* be the value returned by objfun at $x+p$ and calculate the ratio of actual to predicted decrease in the objective function

$$\rho = \frac{f^* - f}{g^T p + \frac{1}{2} p^T B p}$$

If $\rho \geq 1/4$, then we accept $x + p$ as the next iterate. Moreover, if $\rho > 3/4$ and the step was constrained ($p^T D^2 p = r^2$), then we increase the trust region radius to 2 times its current value or rmax, whichever is least, If $\rho < 1/4$, then we do not accept $x + p$ as the next iterate and remain at x. Moreover, we decrease the trust region radius to 1 / 4 of its current value.

The trust region algorithm is known to be highly efficient and very safe. It is guaranteed to converge to a point satisfying the first and second order necessary conditions (gradient is zero and Hessian is positive semidefinite) for a local minimum (Fletcher, 1987, Theorem 5.1.1; Nocedal and Wright, 1999, Theorem 4.8) if the level set of the objective function below the starting position is bounded. If the point to which the algorithm converges actually satisfies the second order sufficient condition (Hessian is positive definite and Lipschitz in a neighborhood of this point), then the algorithm converges at second order (Fletcher, 1987, Theorem 5.1.2).

The algorithm is not designed for use on functions of thousands of variables or for functions for which derivatives are not available. Use nlm or optim for them. It is designed to do the best possible job at local optimization when derivatives are available. It is much safer and much better behaved than nlm or optim. It is especially useful when function evaluations are expensive, since it makes the best possible use of each function, gradient, and Hessian evaluation.

The algorithm is not designed for constrained optimization. It does allow for a restricted domain, but does not converge efficiently to solutions on the boundary of the domain. The theorems mentioned

above assure rapid convergence to a local optimum (at least a point satisfying the first and second order necessary conditions) if the level set of the objective function below the starting position is bounded and is contained in the interior of the domain of the objective function (that is, all points on the boundary of the domain have higher objective function values than the starting point). The algorithm automatically adjusts the trust region to keep accepted iterates in the interior of the domain. This is one way it is safer than `nlm` or `optim`, which do not handle general restricted domains.

Value

A list containing the following components:

`value`	the value returned by `objfun` at the final iterate.
`gradient`	the gradient returned by `objfun` at the final iterate.
`hessian`	the Hessian returned by `objfun` at the final iterate.
`argument`	the final iterate.
`converged`	if `TRUE` the final iterate was deemed optimal by the specified termination criteria.
`iterations`	number of trust region subproblems done (including those whose solutions are not accepted).
`argpath`	(if `blather == TRUE`) the sequence of iterates, not including the final iterate.
`argtry`	(if `blather == TRUE`) the sequence of solutions of the trust region subproblem.
`steptype`	(if `blather == TRUE`) the sequence of cases that arise in solutions of the trust region subproblem. `"Newton"` means the Newton step solves the subproblem (lies within the trust region). Other values mean the subproblem solution is constrained. `"easy-easy"` means the eigenvectors corresponding to the minimal eigenvalue of the rescaled Hessian are not all orthogonal to the gradient. The other cases are rarely seen. `"hard-hard"` means the Lagrange multiplier for the trust region constraint is minus the minimal eigenvalue of the rescaled Hessian; `"hard-easy"` means it isn't.
`accept`	(if `blather == TRUE`) indicates which of the sequence of solutions of the trust region subproblem were accepted as the next iterate. (When not accepted the trust region radius is reduced, and the previous iterate is kept.)
`r`	(if `blather == TRUE`) the sequence of trust region radii.
`rho`	(if `blather == TRUE`) the sequence of ratios of actual over predicted decrease in the objective function in the trust region subproblem, where predicted means the predicted decrease in the two-term Taylor series model used in the subproblem.
`valpath`	(if `blather == TRUE`) the sequence of objective function values at the iterates.
`valtry`	(if `blather == TRUE`) the sequence of objective function values at the solutions of the trust region subproblem.
`preddiff`	(if `blather == TRUE`) the sequence of predicted differences using the two-term Taylor-series model between the function values at the current iterate and at the solution of the trust region subproblem.
`stepnorm`	(if `blather == TRUE`) the sequence of norms of steps, that is distance between current iterate and proposed new iterate found in the trust region subproblem.

References

Fletcher, R. (1987) *Practical Methods of Optimization*, second edition. John Wiley, Chichester.

Nocedal, J. and Wright, S. J. (1999) *Numerical Optimization*. Springer-Verlag, New York.

See Also

nlm and optim for competitors that do not require analytical derivatives. deriv to calculate analytical derivatives.

Examples

```
##### Rosenbrock's function #####
objfun <- function(x) {
    stopifnot(is.numeric(x))
    stopifnot(length(x) == 2)
    f <- expression(100 * (x2 - x1^2)^2 + (1 - x1)^2)
    g1 <- D(f, "x1")
    g2 <- D(f, "x2")
    h11 <- D(g1, "x1")
    h12 <- D(g1, "x2")
    h22 <- D(g2, "x2")
    x1 <- x[1]
    x2 <- x[2]
    f <- eval(f)
    g <- c(eval(g1), eval(g2))
    B <- rbind(c(eval(h11), eval(h12)), c(eval(h12), eval(h22)))
    list(value = f, gradient = g, hessian = B)
}

trust(objfun, c(3, 1), 1, 5)

##### function with restricted domain #####
d <- 5
mu <- 10 * seq(1, d)
objfun <- function(x) {
    normxsq <- sum(x^2)
    omnormxsq <- 1 - normxsq
    if (normxsq >= 1) return(list(value = Inf))
    f <- sum(x * mu) - log(omnormxsq)
    g <- mu + 2 * x / omnormxsq
    B <- 4 * outer(x, x) / omnormxsq^2 + 2 * diag(d) / omnormxsq
    list(value = f, gradient = g, hessian = B)
}

whoop <- trust(objfun, rep(0, d), 1, 100, blather = TRUE)
whoop$converged
whoop$iterations
data.frame(type = whoop$steptype, rho = whoop$rho, change = whoop$preddiff,
    accept = whoop$accept, r = whoop$r)

##### solution
```

```
whoop$argument
##### distance of solution from boundary
1 - sqrt(sum(whoop$argument^2))

##### fail when initial point not feasible
## Not run: trust(objfun, rep(0.5, d), 1, 100, blather = TRUE)
```

Index